CLAIMS

1.

A process of validating that a computer program segment with more than one path therethrough is bounded, the computer program segment comprising a recursive cycle or a loop and the process comprising the steps of:

a) assigning a variant ordered array to the cycle or loop, wherein members of the array are expressions derived from functions of variables and/or parameters of the program and the member at each position in the array represents a path through the cycle or loop;

b) defining a predetermined ordered array of corresponding elements of predetermined values;

c) creating a hypothesis that the value of the respective member of the array is decreased when the corresponding path is traversed and the value of the members at earlier positions in the array are unchanged; and that the value of the element of the variant ordered array is never less than the value of the corresponding element of the predetermined ordered array; and

d) proving the hypothesis for each path through the program segment.

2.

A process as claimed in claim 1, wherein the program segment is a method of an object oriented program.

3.

A process as claimed in claim 1, wherein the computer program segment is a loop which may perform any of a plurality of actions dependant on prevailing program conditions, and step a) of assigning a variant ordered array comprises assigning a member of the array to each action such that the value of the member decreases when the corresponding action is performed, and steps c) and d) of creating and proving a hypothesis comprise finding an order of the members of the array such that each action does not change the value of the members of the ordered array preceding the member corresponding to that action.

4.

A process as claimed in claim 1, wherein the program segment is an isolated recursive cycle that does not intersect with any other cycle, but wherein the cycle can call itself in more than one way, and step a) of assigning a variant ordered array comprises assigning a member of the array for each way in which the cycle can call itself, and steps c) and d) of creating and proving a hypothesis comprise finding an order of the members of the array such that way of calling itself does not change the value of the members of the ordered array preceding the member corresponding to that way of calling itself.

5.

A process as claimed in claim 1, wherein the program segment comprises intersecting recursive cycles that intersect solely at a single point and wherein step a) of assigning a variant ordered array comprises assigning a variant ordered array to the point where the recursive cycles intersect, wherein members of the array represent each of the intersecting cycles respectively.

6.

A process as claimed in claim 1, , wherein the program segment comprises recursive cycles that intersect with each other at a first plurality of points, wherein step a) of assigning a variant ordered array comprises assigning a variant ordered array to each intersecting cycle such that each variant has at least a second plurality of members and corresponding members of the second plurality of members of each variant are of the same type so that the values thereof may be compared, and step c) of creating a hypothesis comprises creating a hypothesis that for any path from a first point with a variant to the same point or to a second point with a variant, not passing through a third point with a third variant, the value of the corresponding member of the variant decreases and all preceding members of the arrays remain unchanged.

7.

A process as claimed in claim 6, wherein each of the variant arrays for the program segment has an equal number of members.

8.

A process as claimed in claim 1, wherein the step c) of creating a hypothesis comprises the further steps of:

i) for every program segment of the program, producing a list of all other program segments of the program that directly or indirectly override that program segment;

ii) for every program segment, producing a called list of all other program segments called directly by that program segment and for each member of the called list recording whether the call is statically or dynamically bound, and if statically bound recording the target program segment of the call and if dynamically bound recording the target program segment which is overridden by all other target program segments, recording whether target program segments are variant safe, that is whether they have an associated variant array if they are recursive;

iii) producing a closure list for each program segment by listing the program segments called directly or indirectly by said program segment by associating a provisional closure list with said program segment comprising the called list for that program segment and adding to the provisional closure list the called list for each target program segment of that program segment;

iv) removing the called list from any program segment whose closure list does not include the said program segment, and which are therefore not recursive program segments and removing members from the called lists of the remaining program segments any target program segments the closure lists of which do not include that target program segment and which are therefore not recursive;

v) scanning through the remaining called lists for each program segment and for each member of the called lists not recorded as variant safe checking whether all possible target program segments are variant safe and if so recording the member as variant safe;

vi) generating diagnostic messages for any program segments having members of the associated called list not recorded as variant safe; and

vii) for each program segment that declares or inherits a variant, using the called list to generate a tree of all possible paths starting from that program segment and terminating when a call to a program segment with a variant is reached and for each such possible path formulating a hypothesis whose antecedent is a set of conditions under which the path is entered and whose consequence is that the corresponding member of the variant of a final target program segment in the path has decreased and all earlier members of the variant are unchanged relative to the members of the variant at the beginning of the path.

9.

A process as claimed in claim 8, wherein step ii) includes the further step of removing all entries in every called list which take no part in any recursive cycle and do not lead to a recursive cycle, by removing from the called list of each method all possible targets having empty called lists so that all remaining called list entries take part in recursive cycles.

10.

A process as claimed in claim 8, wherein when the program segment is a segment of an object-oriented program and the program segment is a method, the step a) of assigning a variant comprises the further step, whenever a variant is assigned to a class method, of all methods declared in classes derived from a class that overrides that method, of inheriting the variant and declaring for any overriding method additional variant expressions and appending the additional expressions to the inherited expressions.

11.

A system for validating that a computer program segment with more than one path therethrough is bounded, the program segment comprising a recursive cycle or a loop and the system comprising:

a) means for assigning a variant ordered array to the cycle or loop, wherein members of the array are expressions derived from functions of variables and/or parameters of the program and the member at each position in the array represents a path through the cycle or loop;

b) means for defining a predetermined ordered array of corresponding elements of predetermined values;

c) means for creating a hypothesis that the value of the respective member of the array is decreased when the corresponding path is traversed and the value of the members at earlier positions in the array are unchanged; and that the value of the element of the variant ordered array is never less than the value of the corresponding element of the predetermined ordered array; and d) means for proving the hypothesis for each path through the program segment.

12.

A system as claimed in claim 11, wherein the program segment is a method of an object oriented program.

13.

A system as claimed in claim 11, wherein the computer program segment is a loop which may perform any of a plurality of actions dependant on prevailing program conditions, and the means for assigning a variant ordered array comprises means for assigning a member of the array to each action, such that the value of the member decreases when the corresponding action is performed, and the means for creating and proving a hypothesis comprise means for finding an order of the members of the array such that each action does not change the value of the members of the ordered array preceding the member corresponding to that action.

14.

A system as claimed in claim 11, wherein the program segment is an isolated recursive cycle that does not intersect with any other cycle, but wherein the cycle can call itself in more than one way, and the means for assigning a variant ordered array comprises means for assigning a member of the array for each way in which the cycle can call itself, and the means for creating and proving a hypothesis comprise means for finding an order of the members of the array such that way of calling itself does not change the value of the members of the ordered array preceding the member corresponding to that way of calling itself.

15.

A system as claimed in claim 11, wherein the program segment comprises intersecting recursive cycles that intersect solely at a single point, and wherein the means for assigning a variant ordered array comprises means for assigning a variant ordered array to the point where the recursive cycles intersect, wherein members of the array represent each of the intersecting cycles respectively.

16.

A system as claimed in claim 11, wherein the program segment comprises recursive cycles that intersect with each other at a first plurality of points, wherein the means for assigning a variant ordered array comprises means for assigning a variant ordered array to each intersecting cycle such that each variant has at least a second plurality of members and corresponding members of the second plurality of members of each variant are of the same type so that the values thereof may be compared and the means for creating a hypothesis comprises means for creating a hypothesis that for any path from a first point with a variant to the same point or to a second point with a variant, not passing through a third point with a third variant, the value of the corresponding member of the variant decreases and all preceding members of the arrays remain unchanged.

17.

A system as claimed in claim 16, wherein each of the variant arrays for the program segment has an equal number of members.

18.

A system as claimed in claim 11, wherein the means for creating a hypothesis further comprises:

i) for every program segment of the program, means for producing a list of all other program segments of the program that directly or indirectly override that program segment;

ii) for every program segment, means for producing a called list of all other program segments called directly by that program segment, and for each member of the called list means for recording whether the call is statically or dynamically bound, and if statically bound for recording the target program segment of the call, and if dynamically bound for recording the target program segment which is overridden by all other target program segments, for recording whether target program segments are variant safe, that is whether they have an associated variant array if they are recursive;

iii) means for producing a closure list for each program segment by listing the program segments called directly or indirectly by said program segment by associating a provisional closure list with said program segment comprising the called list for that program segment and adding to the provisional closure list the called list for each target program segment of that program segment;

iv) means for removing the called list from any program segment whose closure list does not include the said program segment, and which are therefore not recursive program segments and for removing members from the called lists of the remaining program segments any target program segments the closure lists of which do not include that target program segment and which are therefore not recursive;

v) means for scanning through the remaining called lists for each program segment and for each member of the called lists not recorded as variant safe, and for checking whether all possible target program segments are variant safe and if so for recording the member as variant safe;

vi) means for generating diagnostic messages for any program segments having members of the associated called list not recorded as variant safe;

vii) for each program segment that declares or inherits a variant, means for using the called list to generate a tree of all possible paths starting from that program segment and terminating when a call to a program segment with a variant is reached and for each such possible path means for formulating a hypothesis whose antecedent is a set of conditions under which the path is entered and whose consequence is that the corresponding member of the variant of a final target program segment in the path has decreased and all earlier members of the variant are unchanged relative to the members of the variant at the beginning of the path.

19.

A system as claimed in claim 18, wherein the means for producing a called list further comprises means for removing all entries in every called list which take no part in any recursive cycle and do not lead to a recursive cycle, by removing from the called list of each method all possible targets having empty called lists so that all remaining called list entries take part in recursive cycles.

20.

A system as claimed in claim 18, wherein when the program segment is a segment of an object-oriented program and the program segment is a method, and the means for assigning a variant further comprises, whenever a variant is assigned to a class method, of all methods declared in classes derived from a class that overrides that method, means for inheriting the variant and declaring for any overriding method additional variant expressions and means for appending the additional expressions to the inherited expressions.